TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2008

MID-YEAR

COMP205 / ENGR202
Software Design and Engineering

Time Allowed: 3 Hours

Instructions: There are 180 possible marks on the exam.
Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.
Non-electronic Foreign language dictionaries are allowed.
Calculators ARE NOT ALLOWED.
No reference material is allowed.

Question	Topic	Marks
1.	Java Fundamentals I	30
2.	Principles of Software Design I	30
3.	Practices of Software Engineering I	30
4.	Java Fundamentals II	30
5.	Practices of Software Engineering II	30
6.	Principles of Software Design II	30
	Total	180

Student ID: .....

Question 1. Java Fundamentals I [30 marks]

(a) Consider the following Java code which is part of an adventure game. It compiles and runs without error.

```
/** A Location represents a single room within the game. */
public class LOCATION {
    public ArrayList<Door> exits = new ArrayList<Door>();
    public ArrayList<I> items = new ArrayList<I>();
}

/** A Door connects two locations, and has a front and back side. */
public class Door {
    private LOCATION sideA, sideB;

    /** Construct Door */
    public Door(LOCATION sideA, LOCATION sideB) {
        this.sideA = sideA;
        this.sideB = sideB;
    }

    /** Get location on side A of door */
    public LOCATION getSideA() { return sideA; }
    /** Get location on side B of door */
    public LOCATION getSideB() { return sideB; }
}

/** Represents an item (e.g. furniture, coin, etc.) in the game. */
    public abstract class I {
private String name; // name
    private LOCATION location; // location containing this item

    /** Construct Item */
    public I(String zzz, LOCATION yyy) {
        this.name = zzz; // assign zzz to name field
        this.location = yyy; // assign zzz to name field
    }

    /** This method returns the items name. */
public String getName() { return name; }
    /** This method returns the items name. */
         public LOCATION getLocation(){return location;}
}

public class Coin extends I {
    public int value;
    public Coin(int value, String name, LOCATION location) {
        super(name,location);this.value=value;
}   }
```

Student ID: . . . . . .

(i) [10 marks] Circle and number five separate problems of style with the four classes on the previous page. For each problem, write a brief (i.e. one or two line) description of the problem in the corresponding box below.

1)

2)

3)

4)

5)

(ii) [8 marks] In the box below, draw a *UML Class Diagram* for the classes shown on the previous page. (Show relationships, multiplicity and class attributes in your diagram, but not methods).

(b) Consider the following Java code, which compiles without error.

```
public boolean checkValid(LOCATION room) {
   // Look through exits of this room
   for(Door d : room.exits) {
        LOCATION next = d.getSideA();

        // which side doesn't this room lead to?
        if(next == room) { next = d.getSideB(); }
        // is the door there?
        if(!next.exits.contains(d)) { return false; }
        // now look through doors in next room
        if(!checkValid(next)) { return false; }
    }
    // Yes, it's all ok!
    return true;
}
```

(i) [7 marks] In the box below, briefly describe in your own words what this method does.

(ii) [5 marks] The method above contains a serious bug. Briefly describe what the problem is, and under what circumstances it occurs.

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: . . . . . .

Question 2. Principles of Software Design I [30 marks]

(a) Consider the following specification for a music player.

> "A music player plays music when requested. A user can start, stop and pause playback, as well as fast-forward/rewind the current track. A *simple* music player is either a *cassette*, *CD* or *MP3* player. A *combination* music player consists of one or more simple music players (e.g. a cassette *and* CD player). "

(i) [2 marks] According to the specification, what *interface* does a music player have?

(ii) [2 marks] From the specification, what do you believe constitutes the *state* of a simple music player?

(iii) [2 marks] How would you justify the claim that a music player has *identity*?

(iv) [2 marks] According to the specification, what *inheritance* relationship exists amongst the different kinds of music player?

(v) [2 marks] What *Design Pattern* could be used to implement combination music players?

(b) Consider the following Java code, which compiles and runs without error.

```
public class SortedIntList {
  public ArrayList<Integer> ints = new ArrayList<Integer>();

  public void add(int x) { ints.add(x); Collections.sort(ints); }
  public void remove(int x) { ints.remove(x); }
}
```

(i) [2 marks] In the box below, give a Java method which accepts a `SortedIntList` and prints out its elements in order.

(ii) [4 marks] The `SortedIntList` class is not properly *encapsulated*. Identify two distinct ways a user can put a `SortedIntList` into an erroneous state.

(iii) [4 marks] Suppose you turned the `ints` field of `SortedIntList` into a `LinkedList`. Discuss what changes are necessary to code that uses a `SortedIntList`.

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

(iv) [4 marks] Suppose you made the `ints` field of `SortedIntList` private. Briefly discuss what changes (if any) would be necessary to `SortedIntList`.

(v) [6 marks] Taking into consideration your answers to **(ii)**, **(iii)** and **(iv)**, discuss why *public fields* are considered harmful.

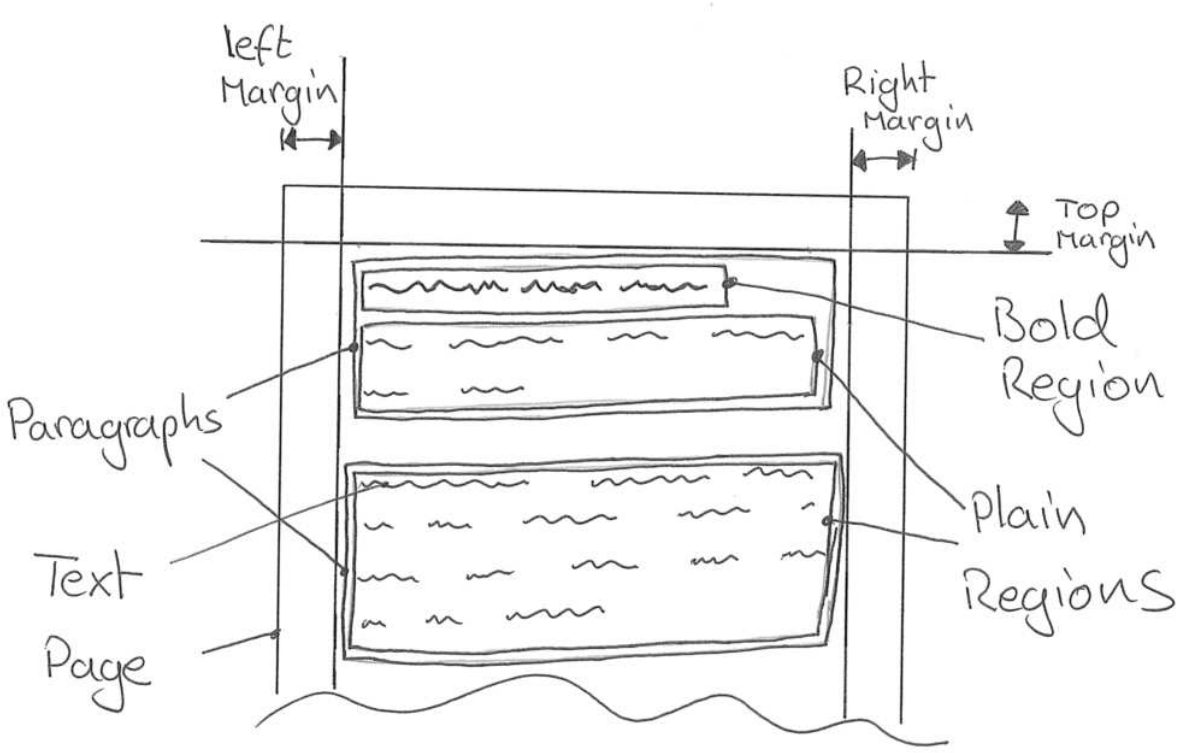
Student ID: .....

Question 3. Practices of Software Engineering I [30 marks]

You are part of a software team developing a word processing application. Your job is to develop the class structure for representing text documents. A rough description of a text document follows:

> "A text document contains one or more *pages* of text, where each page contains zero or more *paragraphs*. Each page has its own *left*, *right*, *top* and *bottom* margins; these measure the distance from the edges of the page.
>
> Paragraphs contain *formatted regions* of characters. Each formatted region is either: a *plain region*; a *bold region*; an *italic* region; or, a *font* region. Plain regions identify regions of text which have no formatting; bold and italic regions identify regions of text to be shown in **bold** or *italic*; finally, font regions identify regions of text to be shown in a particular font (indicated by a string).
>
> Observe that formatted regions may be nested inside other formatted regions. For example, a bold region of text may contain one or more italic regions inside it."

The diagram below illustrates an example text document.



(a) [15 marks] In the box below, draw a *UML Class Diagram* that uses the *composite pattern* to represent text documents and their formatted regions. (Show relationships, multiplicity and class attributes in your diagram, but not methods).

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

(b) [15 marks] In the box below, give Java code for the classes in your design that represent *pages*, *paragraphs* and *bold regions*. In doing this, you should follow good software engineering practice.

Student ID: .....

Question 4. Java Fundamentals II [30 marks]

Consider the following code, **which contains a range of type errors and, hence, does not compile correctly**. Some code has been omitted and is indicated by '...'; you can assume the missing code is not responsible for any compilation problems.

```
public interface Shape { public boolean contains(int x, int y); }

public class Circle implements Shape {
  private Point midPoint; // mid point of circle
  private int radius;

  public Circle(Point mp, int r) { midPoint = mp; radius = mp; }

  public int contains(int x, int y) { ... }
  public int midPoint() { return midPoint; }
  public int radius() { return radius; }
}

public class Square implements Shape {
  private Point[] points = new int[4]; // four corners of square

  public Square(Point p1, Point p2, Point p3, Point p4) { ... }

  public Point[] getPoints() { return points; }
}

public class Main {
  public String check(double x, double y, List<Square> shapes) {
    for(Shape s : shapes) {
      if(!s.contains(x,y)) { return false; }
    }
    return true;
  }

  public static void main(String[] args) {
    List<Circle> circles = new ArrayList<Circle>();
    circles.add(new Circle(new Point(10,10),0));
    circles.add(new Square(new Point(20,20), new Point(30,"20"),
                           new Point(30,30), new Point(20,30)));
    double x1 = 1.0, y1 = 2.0;
    check(x1, y1, circles);

    int x2 = 1, y2 = 2;
    check(x2, y2, new ArrayList<Square>());
} }
```

Student ID: .....

(a) [12 marks] Circle and number six separate problems with the code on the previous page that prevent it from compiling. For each problem, write a brief (i.e. one or two line) description of the problem in the corresponding box below.

1)

2)

3)

4)

5)

6)

(b) [5 marks] Suppose there are two classes, `X` and `Y`, where `Y` is known to be a *subtype* of `X` (i.e. `X` $\geq$ `Y`). What can you tell about the relationship between `X` and `Y`?

(c) [3 marks] Consider the last line on the previous page, which reads "`check(x2,y2,new ArrayList<Shape>())`". **This line contains no errors**. Therefore, what can you tell about the relationship between types `double` and `int`?

(d) [10 marks] The following `NonNullList` class implements a list which cannot hold `null`. Carefully and neatly make this class into a generic `NonNullList`, so that it can store any kind of object.

```
public class NonNullList implements Iterable {

  private List list;

  public NonNullList(List list) { this.list = list; }

  public NonNullList() { list = new ArrayList(); }

  public boolean add(Object x) {

    if(x == null) {

     return false;

    } else {

      return list.add(x);

    }

  }

  public boolean remove(Object x) {

    if(x == null) {

     return false;

    } else {

      return list.remove(x);
    }

  }

  public Object get(int index) { return list.get(index); }

  public int size() { return list.size(); }

  public Iterator iterator() { return list.iterator(); }
}
```

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 5. Practices of Software Engineering II [30 marks]

Consider the following Java classes which compile correctly, and the outline of a JUnit class for testing them given below:

```
public class Competitor {
  final String name;
  final int time;
  Competitor(String n, int t) {name = n; time = t; }
}

public class TopThree {
  private Competitor gold, silver, bronze;
  public Competitor getGold() { return gold;}
  public Competitor getSilver() { return silver;}
  public Competitor getBronze() { return bronze;}

  public void time(Competitor comp) {
    Competitor c = comp;
    // if no gold competitor, then add one
    if (gold == null) {gold = comp; return;}
    // if time is better (i.e.  faster), then replace gold competitor
    if (c.time < gold.time) { comp = gold; gold = c; c = comp;}
    // if no silver competitor, then add one
    if (silver == null) {silver = comp; return;}
    // if time is better (i.e.  faster), then replace silver competitor
    if (comp.time < silver.time) { comp = silver; silver = c; }
    // if no bronze competitor, then add one
    if (bronze == null) { bronze = comp; return; }
    // if time is better (i.e.  faster), then replace bronze competitor
    if (comp.time > bronze.time) { bronze = comp; }
} }

public class TopThreeTest {
  private Competitor sam, joe, bill, derek;
  private TopThree topThree = new TopThree();

  @Before public void setUp() {
    sam = new Competitor("Speedy Sam",100);
    joe = new Competitor("Jumping Joe", 120);
    bill = new Competitor("Bouncing Bill", 150);
    derek = new Competitor("Daring Derek", 140);

    topThree.time(sam);
    topThree.time(joe);
    topThree.time(bill);
    topThree.time(derek);
  }
  ... // test cases go here
}
```

Student ID: .....

(a) [10 marks] Each of the following JUnit test methods are part of the `TopThreeTest` class. For each, state whether it passes or fails — if it fails, explain why.

1)

```
@Test public void testGold() {
    assertSame(topThree.getGold(),sam);
}
```

2)

```
@Test public void testSilver() {
  assertEquals(topThree.getSilver(),joe);
}
```

3)

```
 @Test public void testBronze() {
    assertSame(topThree.getBronze(),derek);
 }
```

4)

```
@Test public void testGoldReversed() {
    TopThree tt = new TopThree();
    tt.time(derek);
    tt.time(joe);
    tt.time(bill);
    tt.time(sam);
    assertSame(tt.getGold(),sam);
}
```

5)

```
@Test(expected= NullPointerException.class)
public void testError() {
    TopThree tt = new TopThree();
    assertNull(tt.getGold());
}
```

Student ID: .....

(b) Consider the following Java class which compiles and runs correctly, and the outline of a JUnit class for testing it given below:

```
public class SimpleStack<E> {
  private List<E> rep = new ArrayList<E>();

  public void push(E e) {rep.add(0, e); }
  public E pop() {return rep.remove(0); }
  public boolean isEmpty() {return rep.isEmpty(); }
}

public class SimpleStackTest {
  private SimpleStack<String> sss;

  @Before public void setUp() { sss = new SimpleStack<String>(); }

  ... // tests cases go here
}
```

(i) [15 marks] Write JUnit test methods that could be added into this class to test the following:

1. That a new stack is empty.
2. That a stack with something pushed onto it is not empty.
3. That if you push something onto a new stack, then pop it off again, the stack is still empty.
4. That you can push "null" onto a stack of Strings.
5. That you can push three integers onto a stack of integers, and they are popped off in reverse order.

1)

2)

3)

4)

5)

(ii) [5 marks] Explain why you should write tests **BEFORE** you write method code.

Student ID: .....

Question 6. Principles of Software Design II [30 marks]

This question is about the DataVault interface listed below:

```
/** A DataVault stores data safely.  The data can be any kind of Object
 * and is associated with a String via store/get (like a Hashtable). Data
 * in the Vault is stored in a series of snapshots (copies of the
 * DataVault).  snapshot() makes a new snapshot, while revertToSnapshot()
 * takes the DataVault back to an earlier snapshot.  Finally, dump()
 * saves memory by deleting old snapshots that are still stored, and can
 * be used to get a copy of all data in the vault.  The DataVault can
 * backup data to disk.
 */
public interface DataVault {
  /** Stores new (name,value) pair in the vault for current snapshot.
   * @param propertyName - name where the value is to be stored
   * @param value - what is stored at that name */
  void storeProperty(String propertyName, Object value);

  /** Return the current value of a property from the current snapshot
   * @param name - name that will be looked up in the vault
   * @return value stored with that name, or "MISSING" if not there. */
  Object getProp(String name);

  /** Make a new snapshot and return the id number of that snapshot.
   * (Calling snapshot twice writes contents of the vault to disk)
   * @return the number of the new snapshot */
  int snapshot();

  /** Make the contents of the Vault go back to an earlier snapshot.
   * @param snapshot - the number of the snapshot to go back to
   * @return true if the snapshot exists, false if it doesnt */
  boolean revertToSnapshot(int snapshot);

  /** This method deletes old snapshots.
   * @param i - delete all snapshots before i
   * @return true if it worked */
  boolean dump(int i);

  /** This dumps all information out of the vault.
   * @return Object array containing all the information. Even numbered
   * entries in this array (0,2,4) have an Integer which is a snapshot
   * number. Odd numbered entries are: "DELETED" if that snapshot number
   * is missing (i.e. has been deleted); "DISK" if that snapshot is only
   * on disk; or a Hashtable of (name,value) pairs for a valid snapshot.
   * Warning: be very very careful if you update this array / Hashtable.
   * You might break the whole Vault */
   Object[]  dump(); /* return all information in vault. May be slow */
}
```

Student ID: . . . . . .

(a) [12 marks] Circle and number six separate problems with the design of the DataVault interface on the previous page. For each problem, write a brief (i.e. one or two line) description of the problem in the corresponding box below.

1)

2)

3)

4)

5)

6)

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

(b) [18 marks] In the box below, write a Java interface and JavaDoc comments for your own replacement DataVault specification.

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: . . . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

****